Challenge 2

CY6740 – Machine Learning in Cyber-Security

Name: Tejas Krishna Reddy

NUID: 001423166

<u>Solution:</u>

We have a dataset, which has 11 features and one predictor label with 2 classes. Our objective is to classify the URL data into Malicious URL's or Benign URL's. Typically making this a binary classification problem.

Below is a Pseudo Algorithm:

1. Read the csv data into a Pandas Dataframe which is easy to manipulate and handle.
2. Now, replace the missing values with 0, since all features should have integers while it's being modeled.
3. Divide the data into features 'X' and label 'Y'.
4. Now, we see that the features have a wide scale. For example, 'CONTENT_LENGTH' has integers ranging from 0 to 649263, which is wide range but features such as 'DNS_QUERY_TIME' ranges only from 0 to 20. ML models work best when all its features are in a given range, and in a smaller magnitude. Hence, we take the training features 'X' and scale them using sklearn's standardScaler module.
5. Now, as instructed in the assignment PDF, we divide the X and y, into 80% (for training) and 20% (for testing). While we divide the data, we reshuffle the whole dataset and then divide into 80-20 ratio, in order to avoid bias. By doing this we would have 1424 samples of training data and 357 samples to test it on.
6. We define a KNN Classifier Model, with 3 neighbors. And fit that model into the training data.
7. We then check both the training accuracy and testing accuracy and make sure they are almost equal.
8. We check the confusion matrix, accuracy and precision to understand the sanity of the model.

Results:

The training and testing accuracy after tweaking the hyper-parameters were as follows:

```
from sklearn.metrics import accuracy_score

### Training Accuracy
y_pred_train = modl1.predict(X_train)
print('Training Accuracy = ', accuracy_score(y_train, y_pred_train))


### Testing Accuracy
y_pred_test = modl1.predict(X_test)
print('Testing Accuracy = ', accuracy_score(y_test, y_pred_test))
```

```
Training Accuracy =  0.9648876404494382
Testing Accuracy =  0.9327731092436975
```

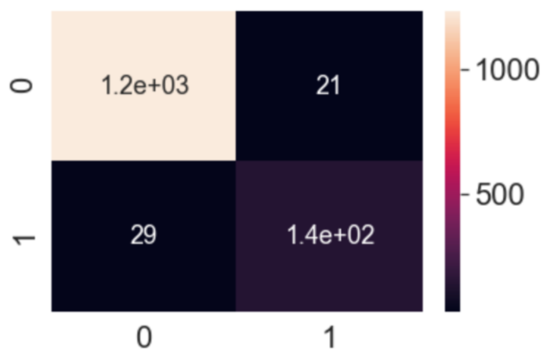
The final precision scores for training and testing datas were as follows:

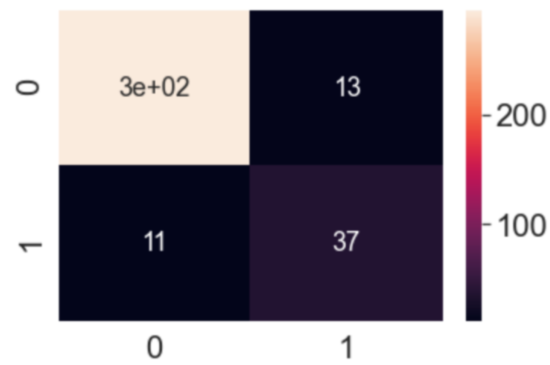
```
from sklearn.metrics import precision_score

### Training Accuracy
y_pred_train = modl1.predict(X_train)
print('Training Precision = ', precision_score(y_train, y_pred_train))


### Testing Accuracy
y_pred_test = modl1.predict(X_test)
print('Testing Precision = ', precision_score(y_test, y_pred_test))
```

```
Training Precision =  0.86875
Testing Precision =  0.74
```

Now let us look at the confusion matrices for test and train datasets.

Training Confusion Matrix:






```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train, modl1.predict(X_train))
```

```
]: array([[1235,   21],
          [  29,  139]], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, modl1.predict(X_test))
```

```
2]: array([[296,  13],
          [ 11,  37]], dtype=int64)
```

Since, it is in imbalanced dataset, accuracies would not be the best measure to trust on. It is more important for us to have the model that can identify the malicious URL's. Hence, Precision scores show us how well the model has been performing over the data. There is a difference between the training and testing precision scores which is likely due to the overfitting issue, but after optimizing the parameters to the best, this is what I could achieve. Similarly, confusion matrices are great visualizing tools that help us understand how much of Type 1 and type 2 error exists, know more about the number of false positives and true negetives we have in our resultant model and take the right measures to make the model better later on.

Below is the Jupyter Notebook code execution with results for the above defined problem.

Challenge 2 - Malicious URL Classification using KNN Classifier

Author: Tejas Krishna Reddy

NUID: 001423166

In [ ]:
```
import pandas as pd
import numpy as np
```

In [15]:
```
## Read the csv data into a DataFrame structure
df = pd.read_csv('Dataset_Challenge2.csv')
```

In [16]:
```
## Visualizing the first 5 rows of the data
df.head(5)
```

Out[16]:

	URL_LENGTH	NUMBER_SPECIAL_CHARACTERS	CONTENT_LENGTH	TCP_CONVERSATION_
0	16	7	263.0	
1	16	6	15087.0	
2	16	6	324.0	
3	17	6	162.0	
4	17	6	124140.0	

Fill the missing 'NA' values with 0.

In [17]:
```
df = df.fillna(0)
```

Seperate the Features and Labels

In [18]:
```
target_variable = 'Type'

X = df.drop([target_variable], 1)
y = df[target_variable]
```

Using Standard Scaling technique to scale all the features.

In [20]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
X = pd.DataFrame(X)
```

Divide the dataset into test (20%) and train (80%).

In [59]:
```
from sklearn import model_selection

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, t
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(1424, 11) (1424,)
(357, 11) (357,)
```

Train the model

In [60]:
```
from sklearn.neighbors import KNeighborsClassifier

modl1 = KNeighborsClassifier(n_neighbors=3, weights='uniform',
                             algorithm='brute', leaf_size=2, p=2, metric='m
```

Training and Testing Accuracy and Precision

In [61]:
```
from sklearn.metrics import accuracy_score

### Training Accuracy
y_pred_train = modl1.predict(X_train)
print('Training Accuracy = ', accuracy_score(y_train, y_pred_train))


### Testing Accuracy
y_pred_test = modl1.predict(X_test)
print('Testing Accuracy = ', accuracy_score(y_test, y_pred_test))
```

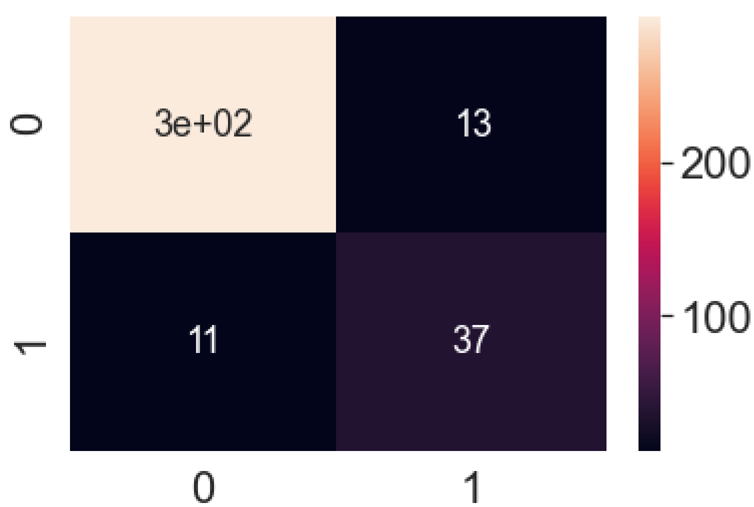
```
Training Accuracy =  0.9648876404494382
Testing Accuracy =  0.9327731092436975
```

In [62]:
```
from sklearn.metrics import precision_score

### Training Accuracy
y_pred_train = modl1.predict(X_train)
print('Training Precision = ', precision_score(y_train, y_pred_train))


### Testing Accuracy
y_pred_test = modl1.predict(X_test)
print('Testing Precision = ', precision_score(y_test, y_pred_test))
```

```
Training Precision =  0.86875
Testing Precision =  0.74
```

In [71]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics

array = (metrics.confusion_matrix(y_test, modl1.predict(X_test)))    ## con

df_cm = pd.DataFrame(array, range(2), range(2))

sns.set(font_scale=2)    #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 18})## font size inside t
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x238596c8d68>



In [72]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, modl1.predict(X_test))
```

Out[72]:
```
array([[296,  13],
       [ 11,  37]], dtype=int64)
```

In [ ]: